



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/720,963	11/24/2003	Richard D. Dettinger	ROC920030278US1	5212
46797	7590	08/22/2007	EXAMINER	
IBM CORPORATION, INTELLECTUAL PROPERTY LAW DEPT 917, BLDG. 006-1 3605 HIGHWAY 52 NORTH ROCHESTER, MN 55901-7829			DWIVEDI, MAHESH H	
		ART UNIT	PAPER NUMBER	
		2168		
		MAIL DATE	DELIVERY MODE	
		08/22/2007	PAPER	

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.	10/720,963	
Examiner	Art Unit	
Mahesh H. Dwivedi	2168	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) Responsive to communication(s) filed on 06 June 2007.

2a) This action is FINAL. 2b) This action is non-final.

3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4) Claim(s) 1-26 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5) Claim(s) _____ is/are allowed.

6) Claim(s) 1-26 is/are rejected.

7) Claim(s) _____ is/are objected to.

8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9) The specification is objected to by the Examiner.

10) The drawing(s) filed on 10 August 2006 is/are: a) accepted or b) objected to by the Examiner. Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a). Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a) All b) Some * c) None of:
1. Certified copies of the priority documents have been received.
2. Certified copies of the priority documents have been received in Application No. _____.
3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) Notice of References Cited (PTO-892) 4) Interview Summary (PTO-413)
2) Notice of Draftsperson's Patent Drawing Review (PTO-948) Paper No(s)/Mail Date. _____.
3) Information Disclosure Statement(s) (PTO/SB/08) 5) Notice of Informal Patent Application
Paper No(s)/Mail Date _____.
6) Other: _____

DETAILED ACTION

Remarks

1. Receipt of Applicant's Amendment, filed on 06/06/2007, is acknowledged. The amendment includes the amending of claims 1, 11, and 15-20.

Specification

2. The disclosure is objected to because of the following informalities: Attorney Docket Number at paragraphs 1 and 46 should be replaced with the Application serial number and its current status

Claim Rejections - 35 USC § 101

3. The rejections raised in the office action mailed on 03/06/2007 have been overcome by the applicant's amendments received on 06/06/2007.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

6. Claims 1-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Amro et al.** (U.S. Patent 6,041,326) in view of **Young** (U.S. Patent 6,560,606).

7. Regarding claim 1, **Amro** teaches a method comprising:

- A) providing an interface for specifying the functional module (Column 10, lines 20-35);

- B) providing a configuration file containing information regarding invocation of the functional module (Column 10, lines 20-35, and lines 52-60);
- C) wherein the configuration file specifies at least an input field of the query result required by the at least one of the functional modules and at least one output field produced by one of the plurality of functional modules (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- D) receiving the query result retrieved from the database (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- E) invoking the functional module to process the query result in a manner determined according to information retrieved from the configuration file (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2); and
- F) returning the processed query result to the application (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2).

The examiner notes that **Amro** teaches “**providing an interface for specifying the functional module**” as “As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism” (Column 10, lines 20-35). The examiner further notes that **Amro** teaches “**providing a configuration file containing information**

regarding invocation of the functional module" as "As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism" (Column 10, lines 20-35) and "The type of plug-in program utilized herein depends upon the desires of a user. The user determines how many predefined traits are to be utilized during a search. For example, a user may be knowledgeable about a certain topic, and may want to avoid certain sites or areas altogether. Depending upon the type of plug-in program designated by the user, the search engine calls this plug-in program to perform the search, avoiding sites or areas in response to instructions processed by the plug-in program" (Column 10, lines 52-60). The examiner further notes that Amro teaches "wherein the configuration file specifies at least an input field of the query result required by the at least one of the functional modules and at least one output field produced by one of the plurality of functional modules" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has

a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "**receiving the query result retrieved from the database**" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the

search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that Amro teaches "invoking the functional module to process the query result in a manner determined according to information retrieved from the configuration file" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and

determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "returning the processed query result to the application" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as

"plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2).

Amro does not explicitly teach:

G) plurality of functional modules.

Young, however, teaches "**plurality of functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing

the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 2, **Amro** further teaches a method comprising:

- A) wherein the interface is a graphic user interface utilized by users to specify a functional module (Column 10, lines 20-35).

The examiner notes that **Amro** teaches "**wherein the interface is a graphic user interface utilized by users to specify a functional module**" as "As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would

be appropriate to handle a customization/selection mechanism" (Column 10, lines 20-35).

Amro does not explicitly teach:

B) functional modules.

Young, however, teaches "**functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 3, **Amro** further teaches a method comprising:

A) wherein the interface allows an external application to specify a functional module (Column 10, lines 7-20).

The examiner notes that **Amro** teaches “**wherein the interface allows an external application to specify a functional module**” as “As described at block 146, a plug-in program is designated for limiting the scope of on-line data searches to particular searches or to certain conditions (e.g., time conditions) such that the plug-in program may be called for processing by the on-line search engine during on-line data searches by the on-line search engine. A calling sequence is utilized to manage the link between the plug-in program and the search engine application. When a plug-in program call occurs, the calling sequence acts as an agreement between the calling routine (i.e., the search engine) and the called plug-in program on how arguments will be passed and in what order, how values will be returned, and which routine will handle any necessary housekeeping (e.g., cleaning up the stack)” (Column 10, lines 7-20).

Amro does not explicitly teach:

B) functional modules.

Young, however, teaches “**functional modules**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24) and “The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing

the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 4, **Amro** does not explicitly teach a method comprising:

- A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules; and
- B) the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules.

Young, however, teaches "**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order

when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6), and "**the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage

in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 5, **Amro** does not explicitly teach a method comprising:
A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed.

Young, however, teaches "**wherein the configuration file contains an explicit sequence in which the functional modules should be executed**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided

for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 6, **Amro** does not explicitly teach a method comprising:

- A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules; and
- B) invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available.

Young, however, teaches "**wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously

(i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60), and "**invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in

reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 7, **Amro** does not explicitly teach a method comprising:

A) wherein invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file comprises invoking at least two functional modules in parallel.

Young, however, teaches “**wherein invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file comprises invoking at least two functional modules in parallel**” as “The system can be implemented to include a multi-stage processing pipeline, each pipeline stage including multiple processing modules and an execution management framework, and capable of multi-threading operation to cause the plug-ins to execute in series or in parallel to speed processing by the plug-ins while accommodating computational dependencies. In some embodiments, the system can also have a metering apparatus for collecting the metered user information, and a presentation manager for providing processed session data to data consumers” (Column 3, lines 22-31) and “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in

reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 8, **Amro** does not explicitly teach a method comprising:

A) wherein the configuration file is in an extensible markup language (XML) format (Column 10, lines 4-7)

Young, however, teaches “**wherein the configuration file is in an extensible markup language (XML) format**” as “The configuration manager 150 generates a configuration file for each stage of the pipeline 518, preferably specifying configuration data in XML format” (Column 10, lines 4-7).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 9, **Amro** further teaches a method comprising:

A) wherein at least one of the functional modules is a plug-in component of the application (Column 12, lines 3-17)

The examiner notes that **Amro** teaches “**wherein at least one of the functional modules is a plug-in component of the application**” as “An alternative preferred embodiment of the present invention involves altering the manner in which the actual search engine actually operates to include the ability to receive directions and instructions directly from the user-defined plug-in program. Instead of merely acting as a filter as described herein, the user-defined plug-in program can function to direct the flow of the search engine while it is searching. Such an implementation requires exposure of API's into the search engine so that a plug-in programmer can control various aspects of the search. This allows the search engine to be controlled programmatically by the plug-in program. Those skilled in the art will appreciate that an API (Application Program Interface) is a set of routines utilized by an application

program to direct the performance of procedures by the computer's operation system" (Column 12, lines 3-17).

Regarding claim 10, **Amro** further teaches a method comprising:

A) wherein at least one of the functional modules is an external application (Column 10, lines 7-35).

The examiner notes that **Amro** teaches "**wherein at least one of the functional modules is an external application**" as "As described at block 146, a plug-in program is designated for limiting the scope of on-line data searches to particular searches or to certain conditions (e.g., time conditions) such that the plug-in program may be called for processing by the on-line search engine during on-line data searches by the on-line search engine. A calling sequence is utilized to manage the link between the plug-in program and the search engine application. When a plug-in program call occurs, the calling sequence acts as an agreement between the calling routine (i.e., the search engine) and the called plug-in program on how arguments will be passed and in what order, how values will be returned, and which routine will handle any necessary housekeeping (e.g., cleaning up the stack). As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism." (Column 10, lines 7-35).

Regarding claim 11, **Amro** teaches a method comprising:

- A) obtaining a set of one or more parameters required for invoking the specified functional module (Column 10, lines 20-35);
- B) wherein at least one of one or more parameters comprises a field of the query result (Column 10, lines 20-35, lines 52-60);
- C) invoking one or more of the specified functional modules whose required parameters are available in a result set collection (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- D) wherein the result set collection is configured to store the query result and an output of the functional module, as the functional module is invoked to process the query result (Column 11, lines 58-67-Column 12, lines 1-2);
- E) obtaining a result set in response to invoking the one or more functional modules (Column 11, lines 58-67-Column 12, lines 1-2);
- F) adding the result set to the result set collection (Column 11, lines 58-67-Column 12, lines 1-2); and
- H) returning the processed query result to the application (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2).

The examiner notes that **Amro** teaches “**obtaining a set of one or more parameters required for invoking the specified functional modules**” as “As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism” (Column 10, lines 20-

35). The examiner further notes that **Amro** teaches "wherein at least one of one or more parameters comprises a field of the query result" as "As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism" (Column 10, lines 20-35) and "The type of plug-in program utilized herein depends upon the desires of a user. The user determines how many predefined traits are to be utilized during a search. For example, a user may be knowledgeable about a certain topic, and may want to avoid certain sites or areas altogether. Depending upon the type of plug-in program designated by the user, the search engine calls this plug-in program to perform the search, avoiding sites or areas in response to instructions processed by the plug-in program" (Column 10, lines 52-60). The examiner further notes that **Amro** teaches "**invoking one or more of the specified functional modules whose required parameters are available in a result set collection**" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11,

lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that Amro teaches "wherein the result set collection is configured to store the query result and an output of each of the plurality of functional modules, as each of the plurality of functional modules is invoked to process the query result" as "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user"

(Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches “**obtaining a result set in response to invoking the one or more functional modules**” as “As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user “plug-in” program acts as a filter by comparing the search engine “hits” with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., “good” hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a “hit” list and ranking of such hits is presented to the user” (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches “**adding the result set to the result set collection**” as “As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user “plug-in” program acts as a filter by comparing the search engine “hits” with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., “good” hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a “hit” list and ranking of such hits is presented to the user” (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches “wherein the configuration file specifies at least an input field of the query result required by the at least one of the functional modules and at least one output field produced by one of the plurality of functional modules” as “A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that

performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "receiving the query result retrieved from the database" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has

a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "**invoking the plurality of functional modules the process the query result in a manner determined according to information retrieved from the configuration file**" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large

number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "returning the processed query result to the application" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the

resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2).

Amro does not explicitly teach:

- G) repeating steps (A)-(D) until all the specified functional modules have been executed; and
- H) plurality of functional modules.

Young, however, teaches "**repeating steps (A)-(D) until all the specified functional modules have been executed**" as "Each plug-in 662, 664, or 666 includes a dependency counter 670A-C (or, alternatively, is associated with a counter stored in memory). Prior to execution, the execution management framework 652 causes the counter for each plug-in to be loaded with a count that indicates the number of plug-ins on which it depends. As a plug-in 662, 664, or 666 completes execution, it notifies the execution management framework 652. A count conditioning mechanism, e.g., the illustrated decrementer 672 in the framework 652, causes the counter 670A-C of any plug-ins dependent on the then executed plug-in to be decremented by a single count. This continues iteratively upon each execution until a predetermined threshold is

reached, e.g., a count of zero, for a particular plug-in. (Alternatively, an incrementer could be used, and incremented to a predetermined threshold.) When the threshold is reached for the counter 670A-C of a particular plug-in 662, 664, 666, that plug-in can be executed because its dependencies have been resolved" (Column 15, lines 3-20), and "**plurality of functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 12, **Amro** further teaches a method comprising:

A) wherein the result set collection comprises results received in response to issuing a query (Column 11, lines 58-67-Column 12, lines 1-2).

The examiner notes that **Young** teaches “**wherein the result set collection comprises results received in response to issuing a query**” as “As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user” (Column 11, lines 59-67-Column 12, lines 1-2).

Regarding claim 13, **Amro** does not explicitly teach a method comprising:

A) wherein the interface is utilized to specify a plurality of functional modules by specifying a multi-analysis functional module.

Young, however, teaches “**wherein the interface is utilized to specify a plurality of functional modules by specifying a multi-analysis functional module**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a

mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 14, **Amro** does not explicitly teach a method comprising:
A) wherein obtaining the set of one or more parameters required for invoking the specified functional modules comprises retrieving information from a configuration file relating the multi-analysis functional module to the specified functional modules.

Young, however, teaches "**wherein obtaining the set of one or more parameters required for invoking the specified functional modules comprises retrieving information from a configuration file relating the multi-analysis functional module to the specified functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring

the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 15, **Amro** teaches a computer readable storage medium comprising:

- A) providing an interface for specifying a functional module (Column 10, lines 20-35);

- B) providing a configuration file containing information regarding invocation of the functional module (Column 10, lines 20-35, and lines 52-60);
- C) wherein the configuration file specifies at least an input field of the query result required by the at least one of the functional modules and at least one output field produced by one of the plurality of functional modules (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- D) receiving the query result retrieved from the database (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- E) invoking the functional module to process the query result in a manner determined according to information retrieved from the configuration file (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2); and
- F) returning the processed query result to the application (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2).

The examiner notes that **Amro** teaches “**providing an interface for specifying a functional module**” as “As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism” (Column 10, lines 20-35). The examiner further notes that **Amro** teaches “**providing a configuration file containing information**

regarding invocation of the functional module" as "As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism" (Column 10, lines 20-35) and "The type of plug-in program utilized herein depends upon the desires of a user. The user determines how many predefined traits are to be utilized during a search. For example, a user may be knowledgeable about a certain topic, and may want to avoid certain sites or areas altogether. Depending upon the type of plug-in program designated by the user, the search engine calls this plug-in program to perform the search, avoiding sites or areas in response to instructions processed by the plug-in program" (Column 10, lines 52-60). The examiner further notes that **Amro** teaches "wherein the configuration file specifies at least an input field of the query result required by the at least one of the functional modules and at least one output field produced by one of the plurality of functional modules" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has

a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "**receiving the query result retrieved from the database**" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the

search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that Amro teaches "invoking the functional module to process the query result in a manner determined according to information retrieved from the configuration file" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and

determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that **Amro** teaches "returning the processed query result to the application" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as

"plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2).

Amro does not explicitly teach:

G) plurality of functional modules.

Young, however, teaches "**plurality of functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing

the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 16, **Amro** does not explicitly teach a computer readable storage medium comprising:

- A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules; and
- B) the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules.

Young, however, teaches "**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in

depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6), and "**the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made

up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 17, **Amro** does not explicitly teach a computer readable storage medium comprising:

A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed.

Young, however, teaches "wherein the configuration file contains an explicit sequence in which the functional modules should be executed" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that

the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 18, **Amro** does not explicitly teach a computer readable storage medium comprising:

- A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules; and
- B) invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available.

Young, however, teaches "wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses

that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60), and "**invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 19, **Amro** does not explicitly teach a computer readable storage medium comprising:

- A) wherein the configuration file is in an extensible markup language (XML) format (Column 10, lines 4-7)

Young, however, teaches “**wherein the configuration file is in an extensible markup language (XML) format**” as “The configuration manager 150 generates a configuration file for each stage of the pipeline 518, preferably specifying configuration data in XML format” (Column 10, lines 4-7).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 20, **Amro** teaches a system comprising:

- A) a processor (Column 5, lines 27-36, Figure 2);
- B) a functional module, wherein the functional module is configured to process a query result retrieved from a database (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2);
- C) a configuration file containing information regarding execution of the functional module (Column 10, lines 20-35, and lines 52-60);
- D) wherein the configuration file specifies at least an input field of the query result required by at least one of the functional modules and at least one output field produced

by one of the plurality of functional modules (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2); and
E) an application from which the functional module is accessible, herein the application which, when executed by a processor, is configured to provide an interface for specifying a functional module and execute the functional module to process the query result in a manner determined according to information retrieved from the configuration file and present a user of the application with the processed query result (Column 10, lines 61-67-Column 11, lines 1-5, Column 11, lines 10-25, Column 11, lines 59-67-Column 12, lines 1-2).

The examiner notes that **Amro** teaches “**a processor**” as “Referring now to FIG. 2 there is depicted a block diagram of selected components in personal computer 10 of FIG. 1 in which a preferred embodiment of the present invention may be implemented. Personal computer 10 of FIG. 1 preferably includes a system bus 20, as depicted in FIG. 2. System bus 20 is utilized for interconnecting and establishing communication between various components in personal computer 10. Microprocessor or central processing unit (CPU) 22 is connected to system bus 20 and also may have numeric co-processor 24 connected to it” (Column 5, lines 27-36). The examiner further notes that **Amro** teaches “**a functional module, wherein the functional module is configured to process a query result retrieved from a database**” as “As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism” (Column 10, lines 20-

35) and "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5). The examiner further notes that **Amro** teaches "**a configuration file containing information regarding execution of the functional module**" as "As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism" (Column 10, lines 20-35) and "The type of plug-in program utilized herein depends upon the desires of a user. The user determines how many predefined traits are to be utilized during a search. For example, a user may be knowledgeable about a certain topic, and may want to avoid certain sites or areas altogether. Depending upon the type of plug-in program designated by the user, the search engine calls this plug-in program to perform the search, avoiding sites or areas in response to instructions processed by the plug-in program" (Column 10, lines 52-60). The examiner further notes that **Amro** teaches "**wherein the configuration file specifies at least an input field of the query result required by at least one of the functional modules and at least one output field**

produced by one of the plurality of functional modules" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2). The examiner further notes that Amro teaches "**an application from which the functional module is accessible, herein the application which, when executed by a processor, is**

configured to provide an interface for specifying a functional module and execute the functional module to process the query result in a manner determined according to information retrieved from the configuration file and present a user of the application with the processed query result" as "A user-defined plug-in program thus functions between the actual search engine utilized by the user and the user. Search results from the search engine can be filtered through such user-defined plug-in programs. The filtered search results are then displayed for the user as the actual search results. Traditional search engines require the user to statically specify the search criteria in advance of performing the actual search. In a preferred embodiment of the present invention, however, dynamic search criteria are provided. The search provides an intelligent program that performs dynamic search decisions based on data presented to it during the search" (Column 10, lines 61-67-Column 11, lines 1-5), "If the plant life has a common name that results in a large number of hits resulting from a particular search via a search engine, a program that "plugs" into the search engine applies a series of tests and determinations to the resulting data stream of search engine "hits." Such algorithms are referred to in the art as "plug-in" programs or also as "plug-ins." The plug-in program determines if a given "hit" is linked to the few parts of the "world" in which the university professor knows that these particular types of plant life exist. Aside from the addition of an "AND" condition associated with the remote network location resulting from the data search by the search engine, the URL (Universal Resource Locator) associated with the resulting "hit" is checked by the plug-in program to exclude certain groups that are determined not to be useful to the professor" (Column 11, lines 10-25), and "As illustrated at block 185, the user performs search customizations (i.e., setting parameters) via a Graphical User Interface, assuming such a Graphical User Interface is utilized in association with the search engine. As depicted at block 186, a search is then performed and as depicted at block 187, the user "plug-in" program acts as a filter by comparing the search engine "hits" with the database of known (i.e., previous) hits. Undesirable hits are thus weeded out in this manner, and the desirable hits (i.e., "good" hits) are presented, as illustrated at

block 188. Finally, as illustrated at block 190, a "hit" list and ranking of such hits is presented to the user" (Column 11, lines 59-67-Column 12, lines 1-2).

Amro does not explicitly teach:

F) plurality of functional modules.

Young, however, teaches "**plurality of functional modules**" as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 21, **Amro** further teaches a system comprising:

A) wherein the application is a query building application (Column 10, lines 7-35).

The examiner notes that **Amro** teaches “**wherein the application is a query building application**” as “As described at block 146, a plug-in program is designated for limiting the scope of on-line data searches to particular searches or to certain conditions (e.g., time conditions) such that the plug-in program may be called for processing by the on-line search engine during on-line data searches by the on-line search engine. A calling sequence is utilized to manage the link between the plug-in program and the search engine application. When a plug-in program call occurs, the calling sequence acts as an agreement between the calling routine (i.e., the search engine) and the called plug-in program on how arguments will be passed and in what order, how values will be returned, and which routine will handle any necessary housekeeping (e.g., cleaning up the stack). As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism.” (Column 10, lines 7-35).

Regarding claim 22, **Amro** further teaches a system comprising:

A) wherein at least one of the plurality of functional modules is a plug-in component of the application (Column 12, lines 3-17)

The examiner notes that **Amro** teaches “**wherein at least one of the plurality of functional modules is a plug-in component of the application**” as “An alternative preferred embodiment of the present invention involves altering the manner in which the actual search engine actually operates to include the ability to receive directions and instructions directly from the user-defined plug-in program. Instead of merely acting as a filter as described herein, the user-defined plug-in program can function to direct the flow of the search engine while it is searching. Such an implementation requires exposure of API's into the search engine so that a plug-in programmer can control various aspects of the search. This allows the search engine to be controlled programmatically by the plug-in program. Those skilled in the art will appreciate that an API (Application Program Interface) is a set of routines utilized by an application program to direct the performance of procedures by the computer's operation system” (Column 12, lines 3-17).

Regarding claim 23, **Amro** further teaches a system comprising:

A) wherein at least one of the plurality of functional modules is an external application (Column 10, lines 7-35).

The examiner notes that **Amro** teaches “**wherein at least one of the plurality of functional modules is an external application**” as “As described at block 146, a plug-in program is designated for limiting the scope of on-line data searches to particular searches or to certain conditions (e.g., time conditions) such that the plug-in program may be called for processing by the on-line search engine during on-line data searches by the on-line search engine. A calling sequence is utilized to manage the link between the plug-in program and the search engine application. When a plug-in program call occurs, the calling sequence acts as an agreement between the calling routine (i.e., the search engine) and the called plug-in program on how arguments will be passed and in what order, how values will be returned, and which routine will handle any necessary housekeeping (e.g., cleaning up the stack). As depicted at block 147, predefined parameters to be processed by the plug-in program are designated. Those skilled in the art will appreciate that the operation described at block 147 may be utilized

in accordance with a Graphical User Interface for allowing the user to specify search criteria and customization parameters. Those skilled in the art will also appreciate that such a Graphical User Interface can also present a selection/customization mechanism for the plug-in program. While it is possible to completely embody the customization of the search within the plug-in program, those skilled in the art will appreciate that such an implementation would not be very usable, since altering the search criteria would necessitate recompiling the plug-in program. Thus, a Graphical User Interface utilized in association with the plug-in program would be appropriate to handle a customization/selection mechanism." (Column 10, lines 7-35).

Regarding claim 24, **Amro** does not explicitly teach a system comprising:

- A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules; and
- B) the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules.

Young, however, teaches "**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of

functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6), and "**the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the

right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 25, **Amro** does not explicitly teach a system comprising:
A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed.

Young, however, teaches "**wherein the configuration file contains an explicit sequence in which the functional modules should be executed**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching

Young's would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Regarding claim 26, **Amro** does not explicitly teach a system comprising:

- A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules; and
- B) invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available.

Young, however, teaches “**wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24) and “The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed”

(Column 14, lines 46-60), and “**invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24) and “The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed” (Column 14, lines 46-60).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Young's** would have allowed **Amro's** to provide a method for improving efficiency in reducing overhead associated with processing, as noted by **Young** (Column 2, lines 32-35).

Response to Arguments

8. Applicant's arguments with respect to claims 1-26 have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

9. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

U.S. PGPUB 2004/0073782 issued to **Price et al.** on 15 April 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

U.S. Patent 6,782,531 issued to **Young** on 24 August 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

U.S. PGPUB 2004/0205695 issued to **Fletcher** on 14 August 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

U.S. PGPUB 2004/0010791 issued to **Jain et al.** on 15 January 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

10. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Contact Information

11. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Mahesh Dwivedi whose telephone number is (571) 272-2731. The examiner can normally be reached on Monday to Friday 8:20 am – 4:40 pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tim Vo can be reached (571) 272-3642. The fax number for the organization where this application or proceeding is assigned is (571) 273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Mahesh Dwivedi
Patent Examiner
Art Unit 2168


August 17, 2007


TIM VO
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100